Coding Practices

Robert C. Seacord, Software Engineering Institute [vita[3]]
Daniel Plakosh, Software Engineering Institute [vita[4]]

2006-01-04

Most software vulnerabilities are the result of small but reoccurring programming errors that could be easily avoided if programmers learned to recognize them and understand their potential harm. In particular, the C and C++ programming languages have proved highly susceptible to these classes of errors. This knowledge area of the Build Security In web site describes coding practices that can be used to mitigate against these common problems in C and C++.

Most of the documents in this knowledge area are excerpted from the CERT book Secure Coding in C and C++[5] [1], written by Robert C. Seacord with contributions from other members of the CERT Coordination Center. The mitigation strategies included in this knowledge area deal primarily with vulnerabilities resulting from programming errors in string manipulation, integer operations, and dynamic memory management. For a more complete description of common programming errors and the resulting vulnerabilities, please see Secure Coding in C and C++[6].

Secure coding requires an understanding of common programming errors that lead to software vulnerabilities and the knowledge and use of alternative approaches that are less error prone. Secure coding can also benefit from the proper use of software development tools, including compilers. Compilers typically have options that allow increased or specific diagnostics to be performed on code during compilation. Resolving these warnings (by correcting the problem or determining that the warning is superfluous) can improve the security of your deployed software system. Compilers can also provide options that influence runtime settings, such as the /GS flag in Microsoft Visual Studio. Understanding available compiler options and making informed decisions about which options to use and which to omit can help eliminate vulnerabilities and mitigate against runtime exploitation of undiscovered or unresolved vulnerabilities. An example of the use of compiler checks to mitigate against integer vulnerabilities is described in Compiler Checks[7]. Examples of using other static and dynamic analysis tools to discover and mitigate vulnerabilities are described in Runtime Analysis Tools[8] and Heap Integrity Detection[9].

Mitigation strategies are described, including security, performance, availability, ease of use, and other known quality attributes. We do not attempt to describe the conditions under which one mitigation strategy is preferred to another. Instead, we assume that you (the customer of the information) know what your requirements and constraints are and can make an appropriate selection based on your analysis of this information and the information contained in the referenced resources.

String Manipulation

- C++ std::string[11]
- fgets() and gets_s()[12]
- memcpy_s() and memmove_s()[13]
- Runtime Protection[14]
- SafeStr[15]

3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/274-BSI.html (Seacord, Robert C.)
4. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/268-BSI.html (Plakosh, Daniel)
5. http://www.awprofessional.com/bookstore/product.asp?isbn=0321335724&rl=1
6. http://www.awprofessional.com/bookstore/product.asp?isbn=0321335724&rl=1
7. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/coding/278-BSI.html (Compiler Checks)
8. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/coding/311-BSI.html (Runtime Analysis Tools)
9. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/coding/302-BSI.html (Heap Integrity Detection)

- strcpy_s() and strcat_s()[16]
- strcpy() and strcat()[17]
- strlpy() and strlcat()[18]
- strncpy_s() and strncat_s()[19]
- strncpy() and strncat()[20]
- Strsafe.h[21]
- Vstr[22]

Dynamic Memory Management

- Consistent Memory Management Conventions[24]
- Guard Pages[25]
- Heap Integrity Detection[26]
- Null Pointers[27]
- OpenBSD[28]
- Phkmalloc[29]
- Randomization[30]
- Runtime Analysis Tools[31]
- Windows XP SP2[32]

Integers

- Arbitrary Precision Arithmetic[34]
- Compiler Checks[35]
- Range Checking[36]
- Safe Integer Operations[37]
- Strong Typing[38]

Acknowledgments

Documents in this section were authored by Robert C. Seacord and Daniel Plakosh. Documents were reviewed by Shawn Hernan, Michael Howard, and Steve Lipner of Microsoft, Jeffrey Voas of SAIC, and Gary McGraw of Cigital. Editing was performed by Pamela Curtis of the SEI.

References

[1] Seacord, Robert C. *Secure Coding in C and C++*. Boston, MA: Addison Wesley Professional, 2005 (ISBN 0321335724).

Pearson Education, Inc. Copyright